An Enhanced Component Connection Method for Conversion of Fault Trees to Binary Decision Diagrams

R. Remenyte-Prescott; Prof. J.D. Andrews

1. Abstract

Fault Tree Analysis (FTA) is widely applied to assess the failure probability of industrial systems. Many computer packages are available which are based on conventional Kinetic Tree Theory methods. When dealing with large (possibly non-coherent) fault trees, the limitations of the technique in terms of accuracy of the solutions and the efficiency of the processing time becomes apparent. Over recent years the Binary Decision Diagram (BDD) method has been developed that solves fault trees and overcomes the disadvantages of the conventional FTA approach. First of all, a fault tree for a particular system failure mode is constructed and then converted to a BDD for analysis. This paper analyses alternative methods for the fault tree to BDD conversion process.

For most fault tree to BDD conversion approaches the basic events of the fault tree are placed in an ordering. This can dramatically affect the size of the final BDD and the success of qualitative and quantitative analyses of the system. A set of rules are then applied to each gate in the fault tree to generate the BDD. An alternative approach can also be used, where BDD constructs for each of the gate types are first built and then merged to represent a parent gate. A powerful and efficient property, sub-node sharing, is also incorporated in the enhanced method proposed in this paper. Finally a combined approach is developed taking the best features of the alternative methods. The efficiency of the techniques is analysed and discussed.

Keywords: Fault Tree Analysis, Binary Decision Diagrams

2. Introduction

The Binary Decision Diagram (BDD) method [1] has been introduced as a method for efficient and accurate fault tree analysis. This method has been shown to have advantages over the conventional Kinetic Tree Theory [2]. The main strength of the BDD method is the fact that top event probabilities can be calculated without the need to apply approximations or the need to obtain minimal cut sets as intermediate results.

In the BDD method the fault tree is converted to a binary decision diagram, which represents the Boolean logic expression of the particular system failure mode. The method requires to set the variable ordering, and if it is not chosen suitably, the size of the final BDD can grow exponentially. The ordering rules are then applied to construct the BDD (**ite** method [1]). Alternative conversion methods are presented in this paper. These include component connection methods [3] where BDDs for each of the gate types are formed and then joined together according to the type of the parent gate in the fault tree. The basic component connection method is then enhanced by introducing a form of sub-node sharing and then by the development of the hybrid technique utilising the advantageous parts of the component connection method and the **ite** method.

The efficiency of the alternative approaches is evaluated and compared with the conventional **ite** method. Three measures are applied while investigating the suitability of the different techniques. These measures are the final size of the resulting BDD, the number of calculations undertaken and the processing time.

3. Binary decision diagram method

A BDD, shown in Figure 1, is a directed acyclic graph, where all paths through the BDD start at the root vertex and terminate in one of two states – a 1-state (system failure), or a 0-state (system success). The BDD consists of terminal and non-terminal vertices, connected by branches. Every terminal vertex represents the final state of the system and every non-terminal vertex – a basic event of the fault tree. By convention all left branches in the BDD are the 1-branches (component failure occurs), all right branches are the 0-braches (component functions successfully).

The application of the BDD method for system reliability is based on the fact that the BDD encodes the logic function of the system failure in its disjoint form. In the example of a fault tree and its equivalent BDD shown in Figure 2 the logic function is:

$$\text{Top} = a\cdot(b + c)\cdot(b + d) = a\cdot b + a\cdot c\cdot d \qquad [1]$$

where "+" represents Boolean operator OR, "·" represents Boolean operator AND.

There are two possible paths in the BDD shown in Figure 2 that terminate in a system 1 state (failure):

$$a,b \text{ and } a,\bar{b},c,d\ . \qquad [2]$$

Each path is a combination of component states whose existence will result in system failure. Two cut sets can be obtained from these two paths if only the failure events are considered:

$$\{a,b\} \text{ and } \{a,c,d\}. \qquad [3]$$

Cut sets consist only of the vertices that lie on the 1 branches from component nodes in the paths. The cut sets obtained in this example are minimal (they contain necessary and sufficient elements), because the BDD is minimal. Usually it would be necessary to develop a different form of BDD which encodes only the minimal cut sets [1].

In the BDD method the probability of system failure, Q_{SYS}, can be expressed as the sum of the probabilities of the disjoint paths to a terminal 1 in the BDD. This is possible because paths through the BDD are mutually exclusive. The probability of system failure in the example is:

$$Q_{SYS} = q_a q_b + q_a(1-q_b)q_c q_d\ . \qquad [4]$$

A number of other properties including system failure frequency and component importance measures can also be calculated [4].

4. Conventional conversion approach – Rauzy (approach 1)

Rauzy [1] developed a commonly used technique of constructing BDDs. This method applies an **i**f-**t**hen-**e**lse (**ite**) technique to each of the gates in the fault tree. Let $f(\boldsymbol{x})$ be the Boolean function for the top event. Then the given **ite** structure $\textbf{ite}(X, f_1, f_2)$ describes the following situation: **if** variable X occurs (fails) f_1 is considered, **else** f_2 is considered. f_1 and f_2 are Boolean functions, described as the residues of f, with $X = 1$ and $X = 0$ respectively. Therefore, if a node in the BDD encodes variable X, structure f_1 lies below the 1-branch and f_2 lies below the 0-branch of that node.

While applying the **ite** method a variable ordering for basic events is introduced. Then according to the conversion rules every gate in a fault tree is converted to a BDD. The rules are:

If J and H are two events in a fault tree already converted to BDD form and expressed as **ite** structures, i.e. $J = \text{ite}(X, f_1, f_2)$ and $G = \text{ite}(Y, g_1, g_2)$, then for a gate in the fault tree for which these are inputs:

- if X is before Y in the variable ordering ($X < Y$) then

$$J<op>G=\text{ite}(X, f_1<op> G, f_2<op> G) \quad [5]$$

- if $X = Y$ then

$$J<op>G=\text{ite}(X, f_1<op> g_1, f_2<op> g_2) \quad [6]$$

here $<op>$ is the Boolean operator of the gates in the fault tree.

The **ite** technique is explained using the example in Figure 3. The ordering $a < b < c < d < e$ is obtained by traversing the fault tree in a simple top-down left-right way. Applying the connection rules gives the expressions for gates $G1$, $G2$ and Top:

$$\begin{aligned}
G1 &= b \cdot c \cdot d \\
&= \text{ite}(b,1,0) \cdot \text{ite}(c,1,0) \cdot \text{ite}(d,1,0) \\
&= \text{ite}(b,\text{ite}(c,1,0),0) \cdot \text{ite}(d,1,0) \\
&= \text{ite}(b,\text{ite}(c,\text{ite}(d,1,0),0),0) \\
G2 &= b \cdot e \\
&= \text{ite}(b,1,0) \cdot \text{ite}(e,1,0) \\
&= \text{ite}(b,\text{ite}(e,1,0),0) \\
Top &= a + G1 + G2 \\
&= \text{ite}(a,1,0) + \text{ite}(b,\text{ite}(c,\text{ite}(d,1,0),0),0) + G2 \\
&= \text{ite}(a,1,\text{ite}(b,\text{ite}(c,\text{ite}(d,1,0),0),0)) + \\
&\quad \text{ite}(b,\text{ite}(e,1,0),0) \\
&= \text{ite}(a,1,\text{ite}(b,\text{ite}(c,\text{ite}(d,1,\text{ite}(e,1,0)), \text{ite}(e,1,0)),0))
\end{aligned}$$

The resulting BDD is also shown in Figure 3. The **ite** technique produces an ordered BDD, where the variable ordering is retained throughout the BDD. This is observed because every step of the conversion process is performed taking into account the variable ordering.

The **ite** method automatically uses sub-node sharing where each **ite** structure is stored in the memory only once and is reused if calculated further in the process.

5. Component connection methods

5.1. Basic approach (approach 2)

The basic algorithm of the component connection method is explained in [3]. First of all, gates of a fault tree which have only basic events as inputs are considered. Every gate is expressed as a BDD structure for "OR" or "AND" gate types. Then the fault tree structure is ascended considering gates whose inputs have already been expressed as BDDs. A BDD for an "OR" gate or an "AND" gate utilising simple rules of connection is created. Initially BDDs are constructed without considering the repetition of basic events in the fault tree. Then a simplification process for the resulting BDD is performed.

The connection and simplification rules with some alternative strategies are presented in this section. The ordering of basic events is not required for this method, because the connection process

can be performed without following a fixed ordering scheme. Only a selection scheme has to be set which will describe the way in which gate inputs are selected for the connection process.

The *connection rules* are presented below:

1. **For a fault tree gate with only basic events as inputs.** If a gate is an "OR" gate, the BDD nodes representing its inputs are connected to each other through the 0-branches of the nodes. If a gate is an "AND" gate, the BDD nodes are connected through the 1-branches of the nodes (see Figure 4(i) and (ii)).
2. **For a fault tree gate where the inputs are already represented as BDDs.** Then the BDDs are merged to form a BDD of the gate output event. For the two BDDs which represent two inputs to a parent gate, one of them is set to be the main BDD, to which the other will be attached using a rule of selection. Then, if two BDDs are inputs to an "OR" gate, the secondary BDD is connected to every terminal 0-node of the main BDD or if two BDDs are inputs to an "AND" gate, the secondary BDD is connected to every terminal 1-node of the main BDD (see Figure 4(iii) and (iv)).

Using this algorithm the resulting BDD can contain more than one node representing the same basic event on a path. In order to remove repeated events in the BDD each path featuring a repeated variable is simplified using one of the two rules:

1. The first occurrence of an event in the path defines the state of the repeated variable. The node, that represents the second occurrence of the event, needs to be replaced by the events below it on either its 1 or 0 branch. The branch depends on the variable state specified by its first occurrence in the path. For example, if the path passes through the 0-branch of a repeated node, the second appearance of that event should be removed replacing it by the BDD structure below the 0-branch of this second node.
2. If the BDD structures below the 1 and 0 branches of any node are identical, this node is irrelevant and needs to be replaced by the structure below either one of the branches. In other words, if the state of the system does not depend on the occurrence of the basic event, the insignificant node must be removed.

This method has been applied to the fault tree illustrated in Figure 3 resulting in the BDD shown in Figure 5. In this example the fault tree is traversed in the bottom-up manner when constructing a BDD for every gate. The variables are considered in a left-right variable ordering. The left-most BDD input for any gate is set to be the main BDD to which the others are joined.

The conversion process starts constructing two BDDs for gates *G*1 and *G*2, shown in Figure 5(i) and Figure 5(ii) respectively. Gates *G*1 and *G*2 are "AND" gates, therefore, the resulting BDDs are "AND" chains.

Then the top event of the fault tree is considered. The left-most BDD, which represents basic event *a*, is selected to be the main BDD. Then the two BDDs from Figure 5(i) and 5(ii) are connected one by one to the 0 branch of the main BDD, because the top gate is an "OR" gate. The first connection results in the BDD in Figure 5(iii). The BDD after the second connection is presented in Figure 5(iv).

Since there are some repeated events in the final BDD (Figure 5(iv)) the simplification rules are applied. Only one event, *b*, is repeated, therefore, its repetitions need to be removed from three current paths. In the first path *F*1-*F*2-*F*5-*F*6 node *F*5 is replaced by the terminal 0-node. This simplification rule is applied because this path traverses the 0-branch of node *F*2, the first occurrence of the repeated event. In the second path *F*1-*F2*-*F*3-*F*7-*F*8 the repeated event *b* is removed, replacing node *F*7 by node *F*8. In the same way node *F*9 is replaced by node *F*10 in the third path *F*1-*F2*-*F*3-*F*4-*F*9-*F*10. The final BDD is shown in Figure 5(v).

While developing this example no global variable ordering system was used and the basic events were connected according to the order that they appear in the list of gate inputs. However, it is possible to apply a defined ordering scheme for basic events. A number of structural and weighted ordering schemes [5] can be used. The chosen ordering scheme can influence the efficiency of the conversion process.

Using a variable ordering sets the order in which basic events are considered for gates which only feature basic event inputs which are then placed in an "OR" chain or an "AND" chain dependent upon the gate type. When gates are encountered where their inputs have been previously generated it has to be determined how the BDDs will be connected.

During the connection process BDDs were previously selected according to the order that gate inputs are listed, i.e. the BDD, presenting the left-most gate, is set to be the main BDD. Other selection schemes can be used which can result in a smaller BDD and/or in a shorter processing time. Where a global variable ordering scheme is used BDDs can be ordered according to the position of their root vertex in the ordering scheme of basic events. Alternatively selecting according to the smallest number of available branches where connections will be made can offer an advantage in efficiency. Then the efficiency of different strategies can be analysed over a library of different fault tree structures.

The component connection method does not require the introduction of a variable ordering in the conversion process. Therefore, even if the variable ordering is assigned at the start, it is not retained when merging two BDDs. The resulting BDD is not an ordered BDD as it is if obtained using the conventional **ite** approach but it still retains the disjoint path property and can be used for the quantitative analysis.

The most significant disadvantage of the basic component connection method is that it does not use sub-node sharing and this can lead to inefficient memory usage. For example, in Figure 5(v) there are two identical nodes *F*8 and *F*10, which are shared in the BDD obtained using the **ite** technique (Figure 3). Therefore, a form of sub-node sharing is introduced as an extension to the component connection method.

5.2. Sub-node sharing (approach 3)

The sub-node sharing is used in the conventional **ite** technique and provides a significant contribution towards the efficiency of the approach. This property can also be implemented to an extent in the component connection method while two BDDs are connected. Consider for example, merging two inputs for an "OR" gate, as it is shown in Figure 6.

In this example the two BDDs are independent (have no nodes in common), the left BDD is set to be the main BDD. It has two available connection points, i.e. two terminal 0-vertices. They can be connected to the same second BDD. This merging is always suitable since no repeated events appear in the BDDs.

The conversion method of a fault tree to a BDD starts considering those gates which have only basic events as inputs and applying the first connection rule, presented in the previous section. Sub-node sharing is applied while performing the second rule and connecting previously formed BDDs for gate inputs. In this case, when the BDDs contain repeated events the state of each repeated event needs to be considered. During the connection process the secondary BDD can be connected to all appropriate terminal nodes if the path from the root vertex to those terminal nodes has each repeated event in the same state. In other words, the sub-node sharing can be applied if while descending the

BDD from the root vertex the same branches (1-branches or 0-branches) of repeated events were traversed. Otherwise, a new copy of the secondary BDD needs to be used.

The sub-node sharing rule is:

If two paths to terminal vertices (terminal 1-nodes for BDDs being inputs to an AND gate and terminal 0-nodes for BDDs being inputs to an OR gate) fall below the same branches of the repeated events, the same second BDD can be connected to both of the two terminal nodes.

Consider the example from Figure 5, during the last connection of the two BDDs in Figure 5(ii) and in Figure 5(iii), the BDD in 5(iii) is set to be the main BDD and that of Figure 5(ii) the secondary BDD. Since these two BDDs have event *b* in common and represent two gate inputs to an OR gate the paths from the root vertex to the three terminal 0-nodes of the main BDD are investigated. There is only one repeated event *b* in the fault tree. The first path passes the 0-branch of node *b*, the second and the third paths pass the 1-branch of node *b*. The second and the third terminal nodes can be replaced by the same copy of the secondary BDD because the second and the third paths fix the same state for the basic event on the repeated node. The final BDD is shown in Figure 7(i) and 7(ii), after the connection and after the simplification processes respectively.

The resulting BDD in Figure 7(ii) matches the one obtained using the **ite** technique, Figure 3.

It is important to note that when applying the sub-node sharing in the component connection method all repeated events in the system must to be considered, not only those between the two BDDs under the current connection.

5.3. Hybrid approach (approach 4)

This method is introduced to utilise the efficient parts of the two algorithms presented – the **ite** technique and the component connection method. It is clear, that:

i) using the gate constructs for basic events and branches without repeated events BDDs can be immediately formulated without any of processing required by the **ite** method**.**
ii) the sub-node sharing feature of the **ite** method provides a more efficient representation of the logic function than its equivalent introduced in the component connection method.

Therefore, a new method has been created based on the effective features of each approach to obtain the best efficiency for BDD conversion.

As was described before, the variable ordering is not required for the component connection method. However, since the hybrid approach also utilises the **ite** method a variable ordering needs to be introduced. The method then produces ordered BDDs.

While converting a fault tree to a BDD using the hybrid method, a variable ordering needs to be established. Then the building of BDDs for gates containing event inputs only starts, where events are put in a chain according to the type of the gate (the component connection approach). This construction process can be applied regardless of the number of events into a gate without breaking them down into pairs. In comparison, the rules in the **ite** technique deal only with two **ite** structures at once, therefore each gate needs to be preprocessed. The variable ordering needs to be retained while putting basic events in a chain. The comparison of the component connection method and its application in the hybrid method is shown in Figure 8.

Further while building a BDD for a gate, when gate inputs are already represented as BDDs and they do not contain any repeated events, the straightforward connection can be also applied. In this case the variable ordering needs to be applied, i.e. tho BDDs can be merged if all the events of the

main BDD stand before the events of the secondary BDD in the variable ordering. This rule is shown in Figure 9.

Finally, while building a BDD for a gate, when inputs are converted to BDDs and they contain repeated events, the **ite** technique rules are applied.

For example, if we are applying the hybrid method for the conversion of a fault tree in Figure 3, BDDs for gates *G*1 and *G*2 are created using the component connection method, i.e. placing its basic events in "OR" chains as it was shown in Figure 5(i) and (ii). Then the two BDDs are merged applying the **ite** rules, given in equations 5 and 6.

6. Comparison of the methods

The efficiency of each of the methods in the conversion of a fault tree to a BDD depends on the structure of the fault tree. An indication of any advantages of different conversion techniques would need to be determined experimentally and measured over a large range of problems. The four approaches presented in this paper were analysed using a set of 11 fault trees from a benchmark set [6]. Their complexity is indicated in Table 1.

This table describes the complexity of the fault tree in terms of the number of gates, the number of basic events, the number of repeated events and the number of minimal cut sets. In order to obtain a consistent comparison of the four techniques the variable ordering was applied not only for the conventional method but also for the component connection method, i.e. even when it is not needed. The depth-first ordering scheme was applied [5]. In this ordering the left-most gate is always explored completely before considering the remaining gate inputs. The basic events with the greatest number of occurrences are ordered first.

The comparison of the four methods was performed by considering the following measurements:

- the number of nodes in the final BDD,
- the maximum number of lines in the storage array (representing the number of intermediate calculations performed), and
- the processing time.

The results of the three measurements obtained by applying the four methods to the example fault trees are shown in Table 2, Table 3 and Table 4. For the analysis a computer with the 2.16 GHz processor was used.

Remark. Fault trees were simplified [7] prior to BDD conversion process. The simplification process contains the reduction [8] and modularisation [9] techniques. It allows a more efficient analysis of fault trees. The time taken to perform the simplification process has been included in the processing time in Table 4.

The conventional BDD conversion technique (the **ite** method) resulted in smaller BDDs and smaller number of lines in the storage array for all the example fault trees than the basic component connection method. The processing time was also shorter than in the basic component connection method. More results for the basic approach of the component connection method are presented in [10].

When sub-node sharing was introduced in the component connection approach the resulting BDDs were smaller than using the basic component connection method but still larger than from the **ite** technique. Calculation time slightly decreased for the majority of example fault trees. Only for those fault trees with a large number of intermediate calculations (examples 3 and 4) the calculation time

increased, due to an extra time taken while identifying parts in the BDD suitable for the sub-node sharing. Therefore, the total time increased when the sub-node sharing was used. More results on the sub-node sharing method were shown in [11].

The hybrid method resulted in BDDs that contain the same number of nodes as the **ite** approach. Calculation time also remained very similar for all example fault trees. However, the hybrid technique gave slightly better results in terms of the number of lines in the storage array, except example 4. The slightly improved efficiency was due to the fact that the hybrid method used the best rules of the two techniques. Firstly, it was improved because the hybrid method provided the capability to obtain the BDDs for gates with event inputs only in a straightforward way, i.e. putting inputs of a gate in a chain one by one. Secondly, more complex parts of the fault tree were converted to the BDD using the component connection method if repeated events did not appear.

Summarising it is clear that the **ite** technique performs better than the basic component connection method. However, the hybrid method, as a combination of the two methods, can provide the efficiency which is as good as the one of the **ite** method. Despite the fact that the hybrid method is more complex than the **ite** technique, according to the efficiency analysis the hybrid method is a good alternative technique for constructing BDDs.

7. Conclusions

Four methods of constructing BDDs from fault trees were presented in this paper. The first approach is the conventional **ite** technique. The second approach is the basic component connection method. The third approach investigated enhances the basic component connection method by developing a sub-node sharing property. The last method, the hybrid method, utilises the more efficient features of the two basic methods. Some example fault trees were used in order to evaluate the efficiency of the four methods. Three efficiency measures were considered – number of nodes in the final BDD, number of nodes in the storage array and calculation time. It was shown that the **ite** method performs well, whereas the basic component connection method does not provide a good efficiency even if the sub-node sharing is used. When the efficiency of the two techniques was combined and the hybrid method was applied, the efficiency of the conversion method was slightly improved. Therefore, the hybrid can be used as an alternative approach to the **ite** technique for conversion of fault trees to BDDs.

8. References

1. A. Rauzy, "New Algorithms for Fault Tree Analysis," *Reliability Engineering and System Safety*, no. 40, 1993, pp. 203-21.
2. W.E. Vesely, "A Time Dependent Methodology for Fault Tree Evaluation", *Nuclear Design and Engineering*, no. 13, 1970, pp. 337-360.
3. Y.S. Way, D.Y. Hsia, "A simple component-connection method for building binary decision diagrams encoding a fault tree", *Reliability Engineering and System Safety*, no. 70, 2000, pp. 59-70.
4. R.M. Sinnamon, J.D. Andrews, "Improved Accuracy in Quantitative Fault Tree Analysis", *Quality and Reliability Engineering International*, no. 13, 1997, pp. 285-292.
5. K.A. Reay, "Efficient Fault Tree Analysis Using Binary Decision Diagrams", *Doctoral Thesis*, Loughborough University, 2002.
6. A benchmark of Boolean Formulae, http://iml.univ-mrs.fr/~arauzy/aralia/benchmark.html

7. J.D. Andrews, R. Remenyte, “Qualitative Analysis of Complex, Modularised Fault Trees Using Binary Decision Diagrams”, *Proceedings of the IMechE, Part O, Journal of Risk and Reliability*, vol 220, June 2006, pp. 45-53.
8. O. Platz, J.V. Olsen, FAUNET: A program Package for Evaluation of Fault Trees and Networks, *Research Establishment Risk Report*, No. 348, DK-4000 Roskilde, Denmark, 1976.
9. Y. Dutuit, A. Rauzy, A Linear-Time Algorithm to Find Modules of Fault Trees, *IEEE Trans. Reliability*, 45, No.3, 1993, pp422-425.
10. J.D. Andrews, R. Remenyte, "Fault Tree Conversion to Binary Decision Diagrams" , *Proceedings of the 23rd ISSC* , San Diego, USA, August 2005, ISBN 0-9721385-5-2 , [CD-ROM].
11. J.D. Andrews, R.Remenyte, “A Simple Component Connection Approach for Fault Tree Conversion to Binary Decision Diagram”, *Proceedings of the 1st AReS*, Vienna, Austria, April 2006, ISBN 0-7695-2567-9/06.